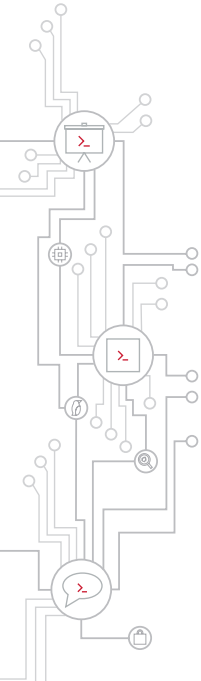


Embedded Linux Security: Dr. Jekyll & Mr. Hyde

Richard Weinberger

2024-09-17

Hello

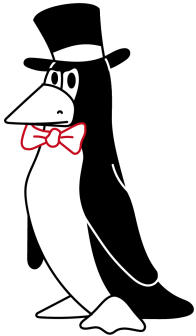
Richard Weinberger:

- Co-founder of sigma star gmbh
- Linux kernel developer and maintainer
- Focus on Linux kernel, low-level components, virtualization, security

$dayjob

- Est. 2011
- Customers all over the world
- IT Services:
  - Engineering
  - Security
  - Training
- Core competences:
  - Embedded Systems
  - Linux Kernel
  - Security Audits
- Contributions to Linux Kernel and other OSS projects

Dr. Jekyll and Mr. Hyde



Why Security for Embedded Devices?

- Compliance with laws
  - GDPR
  - EU Cyber Resiliance Act
  - NIS 1/2
  - Various ISO norms like ISO 62443
- Protection of intellectual property
- Demand from customers (B2C and B2B)
- Hacking corporate networks via embedded systems is real (e.g. printers)

What's The Worst That Can Happen?

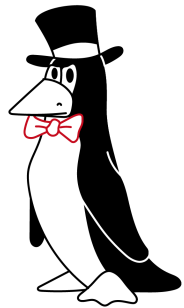
- Various botnets which leverage IoT for DDoS attacks (there have been too many to count now)
- The constant stream of big router+VPN vendor issues (Juniper, Cisco, Forti, ...)
- Remotely hacked cars
- Hacked house hold devices like fridges, washing machines, toasters, ...
- State-sponsored remote backdoor to your device via compression library
- ...

Embedded Linux Today

- Lots of ready-to-use hardware available
- Board support packages are often in good shape
- Huge community
- Lots of fantastic tooling and projects (Yocto/OpenEmbedded, Buildroot, Busybox, Qt)

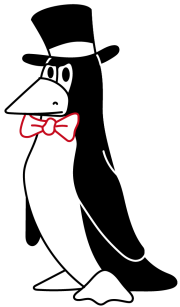
Today's Linux Security Features

- Linux offers everything to lockdown and secure your embedded system
- Basic filesystem permissions
  - Discretionary Access Control (DAC)
  - Access Control Lists (Posix ACLs)
- Advanced filesystem permissions
  - Mandatory Access Control (MAC)
  - Role Based Access Control (RBAC)
  - Multi Level Security (MLS)
- File integrity
  - Integrity Measurement Architecture (IMA)
  - Extended Verification Module (EVM)
  - fs-verity



Today's Linux Security Features in (cont'd)

- Strong encryption *and* authentication for storage
  - dm-crypt
  - dm-verity
  - dm-integrity
- Strong cryptography in kernel and userspace
- Trust/verified boot
- All kinds of randomizations, (k)ASLR, randomized kernel heap, ...
- Various sandboxing technologies
  - seccomp
  - namespaces



What's the Problem?

- You need to use these feature properly
  - Sounds easier than it is
- At best you get a secure platform for free
  - What you do on top of it can poke holes and make the whole system less secure

What's the Problem? (cont'd)

Cause can be innocent-looking technical detail:

- Default compiler flags get overridden
  - you're missing out on security-relevant compiler flags and their warnings
- Application works only with legacy libraries
  - still need to use outdated OpenSSL?
- Application not 64-bits safe
  - you cannot use security-relevant features (NX, ASLR, ...)
- Application running as `root`
  - free privilege escalation for your attackers

Hardware Support

- Many security features depend on hardware features
- Most prominent example: Verified boot
- Lesser known: Disk encryption/authentication
  - Key material needs get stored somewhere
  - In a way nobody can extract it
  - Secure element is needed

Secure Elements

- In the embedded world usually not an TPM
- Something on the SoC, for example:
  - Data CoProcessor (DCP)
  - Cryptographic Acceleration and Assurance Module (CAAM)
- Drivers are often not mainline
  - Yet another maintenance burden
  - Plus a security issue, out-of-tree drivers are often buggy

Permissions

- Your application is fine as long as it runs as `root`
- This might be acceptable initially (e.g. for first PoC), but not after that
- It is often forgotten and then time runs out and you have to ship with everything running as `root`
- This means a single RCE bug will give your attacker free privelege escalation!

Maintenance

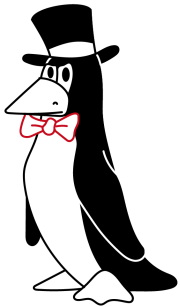
- What if I told you that maintainance is the hardest part?
  - Building the product is just the start
  - Keeping it alive for 5, 10, or more years is the real challenge!

Following Upstream

- Basically two options:
  - Always upgrade to the latest and greatest
  - Backport security fixes yourself
- Both can work, none of them is cheap
- 3rd option: Do nothing
  - Way too popular

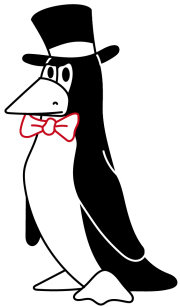
Following Upstream: Always Upgrade

- Let's assume you're using Yocto
  - Regular release: Support for about 8 months
  - LTS release: Support for up to 4 years
- Components *will* get upgraded, API changes
  - Qt, OpenSSL, ...
- You application might need changes
- Full QA cycle needed



Following Upstream: Backport Security Fixes

- You basically fork your Yocto release and keep it yourself alive
- Might work well for a few years if you are keen
- You need to follow all upstream changes
  - How to detect security fixes?
  - How to backport fix for version Y to your older version X?
  - Following just CVEs won't safe you



3rd Option: Do nothing

Elephant in the Room: Linux Kernel

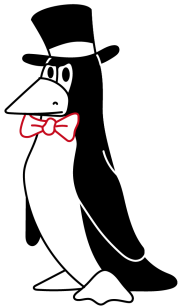
- In many cases you'll run a custom kernel version
  - From your BSP vendor
  - Or you maintain it yourself
  - Seldom the kernel from Yocto or Buildroot
  - Almost never from Debian or other Linux distros
- Depending on the BSP quality, upgrades or even updates can be a challenge
  - Due to out-of-tree drivers
  - Or other changes
  - Security nightmare

Tracking CVEs

- CVEs help a lot, but are far from perfect
- Apply only to supported software
- Finding the fix is not always easy
- Not all projects use CVEs
- Some tooling is available
  - e.g. in Yocto
- Reading all changelogs still needed
  - Yes, I mean it

No Worries, there is Hope!

- Let others do the hard work
  - When possible use something like Debian as base
  - Stick to a recent Yocto release
- Keep your application in shape
  - Try it regularly with recent libraries or base systems
  - Think ahead
- Avoid out-of-tree patches/driver/code as much as possible
  - Being able to use a mainline kernel is awesome!
  - Send patches upstream!
- Don't tie yourself too much to a specific piece of hardware



Summary

- These days Linux offers many mechanisms up secure your embedded system
- It still depends on you
- Don’t underestimate maintenance
- Keep your application(s) in shape, software is a moving target
- Don’t let Mr. Hyde take over!

FIN

Thank you!

Questions, Comments?

Richard Weinberger
richard@sigma-star.at